目录

[一、 C++ 2](#_Toc26161)

[基础 2](#_Toc8769)

[STL 2](#_Toc7740)

[类和对象 3](#_Toc12856)

[二、pyhton 3](#_Toc21803)

[运算相关 3](#_Toc11151)

[list相关 6](#_Toc5852)

[str相关 7](#_Toc17660)

[类相关 7](#_Toc30772)

[其他 9](#_Toc15537)

[三、 基础知识 10](#_Toc20477)

[经典问题 10](#_Toc16965)

[树 11](#_Toc15253)

[列表 12](#_Toc8479)

[图 12](#_Toc30278)

[查找 13](#_Toc14063)

[排序 18](#_Toc70)

[OS 19](#_Toc24379)

[二进制类 19](#_Toc15010)

[Leetcodes 19](#_Toc1035)

# 一、C++

## 基础

1、声明而不定义：extern int i;

声明+定义：int i;

多个文件联合编译时，只有一个文件可以定义，其他文件要用到该变量需要声明

1. 引用必须被初始化，且初始值必须是一个对象（不能是常量），引用不是对象
2. 指针定义的时候是int \*p 而非(int\*)p \*修饰的是p，意是int类型的p指针，如果指针指向了一个对象，就可以用解引用符\*来得到该对象
3. 不能定义指向引用是指针因为引用不是一个对象，但是可以定义指向指针的引用。
4. 对常量的引用也必须是一个常量，反之则不需要，指针也一样，存放常量对象的地址只能通过常量指针，即const可以指向非const，反之不行。

const int i=1;

const int &k=i;

1. cin>>是以空白符（空格、换行、tab）为分隔，而getline(cin,line)是以换行

## STL

### 标准

1、unordered\_set<> 无序哈希表，往里插入相同的数据不会重复，可以用来去重，需include

2、unordered\_map和map

unordered\_map存储机制是哈希表，，即unordered\_map内部元素是无序的。unordered\_map <char, int> T char对应int的字典，map是红黑树，map中的元素是按照二叉搜索树存储，进行中序遍历会得到有序遍历。

3、sort方法，可以对数组，vector这样的用，最后一位不取，自定义的sort方法的使用如下：

bool cmp1(pair<int,int>a,pair<int,int>b)

{

return a.first < b.first;

}

sort(vec.begin(), vec.end(), cmp1);

1. string 的find

string::size\_type position

position=str.find()!=string::npos

1. distance(iter a,iter b)求距离，或者直接迭代器相减
2. find(iter a,iter b,int target)失败返回iter那个类型的end()
3. map的find用法 map.find(type first)
4. string的+，一个常量字符串称为字面值，并不是string对象，ex:”abc”，是不能直接相加，每个加号两边至少有一个是string对象

### vector

1. vector<> 需要include，基本操作函数push\_back()
2. vector的sort用法：sort(vec.begin(),vec.end());

3、vector要修改的话传参数需要传引用&，同样的，对象也是

4、初始化：vector<vector<int>> dp(length1,vector<int>(lenth2,初值))

这个和string有点像 string str(n,str0)

5、vector的一个赋值方法，vec.assign(set.begin(),set.end())

6、非常关键，push\_back会将该容器已经赋值的迭代器弄坏，因此不能对已经使用了迭代器的容器用push\_back等影响容器的操作

7、C++不能定义动态长度的数组，但是可以用二维vector实现相同功能

8、iter->num <====>(\*iter).num 即解引用以后用点

## 类和对象

1、初始化对象用listnode \*node1=new listnode(val)

# 二、pyhton

## 运算相关

1. 两个乘号（\*\*）是乘方
2. 可变对象和不可变对象以及循环赋值问题

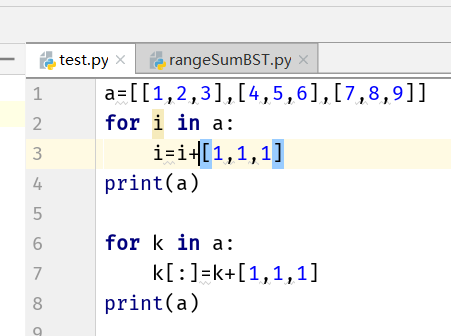
a=b[:] 和a=b 是不一样的

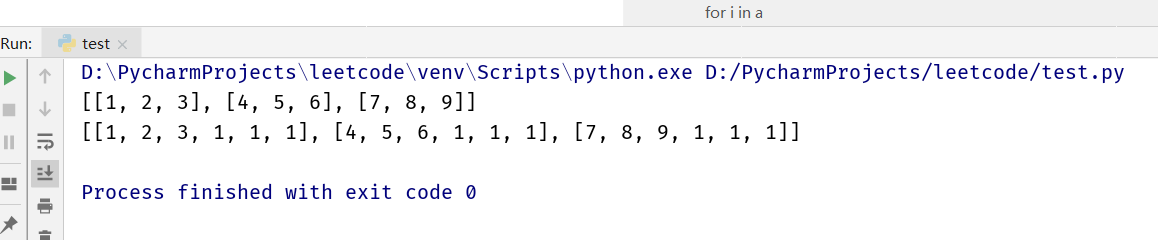
前面那个创建了一个副本 后面那个赋值地址

即应该使用切片赋值列表

也因为这一点，在Python里要是想传递值而不是地址，可以通过赋值切片

For循环中不能直接对list进行修改，需要a[:]=XXX，见图





上面那个图，两个list直接加是append

若是ndarray对象加才是数值的加

然后，list的+=是调用的extend，这个不改变原对象的id

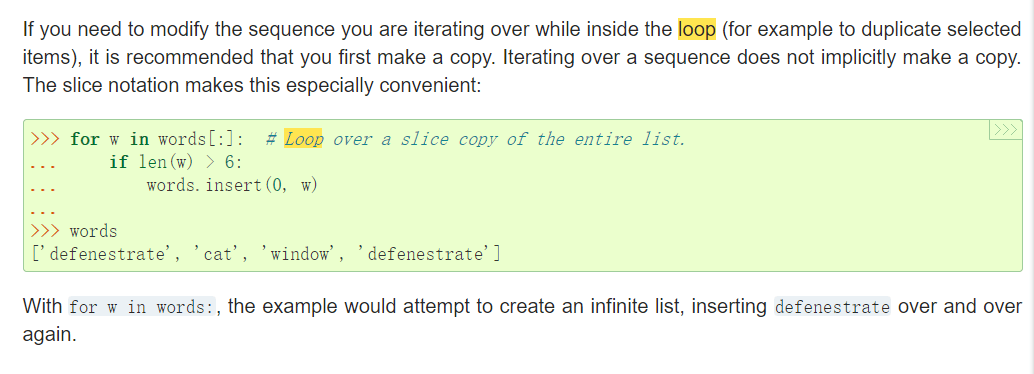
int float str tuple

不可变对象，变量对应内存的值不允许被改变。当变量要改变时，实际上是把原来的值复制一份后再改变，开辟一个新的地址。

List dict set

可变对象





总结：

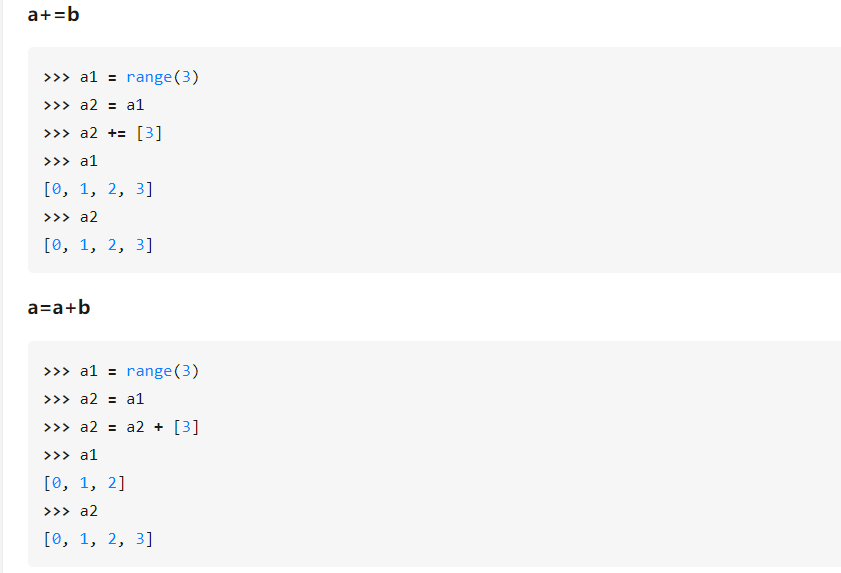
一、对可变对象 +=是不会改地址的，+才会改地址

二、for循环中锁id

三、for循环中对怎么处理？a=a+1=>a+=1 或者 a[:]=a+1

四、用下标做就没问题

3、a?b:c============b if a else c



## list相关

1. list.extend(list a)方法接收一个list做参数，函数的结果将a的每个元素添加到原list后面
2. list.append(a)方法可以接收任何数据类型，并添加到list的尾部。
3. remove(val)只删第一个
4. str.sort(Reverse)是可以带一个bool参数reverse代表顺序，默认为false即正序
5. sort和sorted的区别

a.sort()是方法，a调用sort方法改变a

sorted(list a)是函数，不会改变a

1. for i,j in enumerate(list)

其中i是下标j是值

7、list.index会返回list中出现的所有位置，str的则只会一个

## str相关

1、str.split(‘a’,num)以a分割str最多num次

2、str.strip()=str.strip(‘’)，若只要去掉左或右用lstrip()和rstrip()

3、str.replace(‘a’,’b’)用b替换str中的a

## 类相关

1、class ElectricCar(Car):

"""电动汽车的独特之处"""

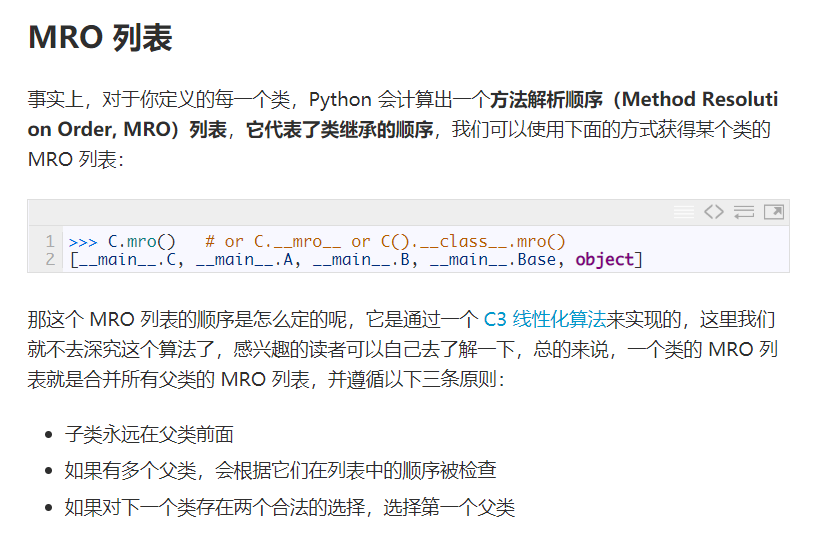
def \_\_init\_\_(self, make, model, year):

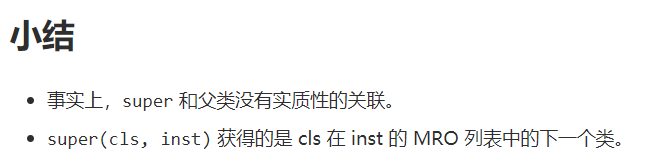
"""初始化父类的属性"""

super().\_\_init\_\_(make, model, year)

2、类里面定义的函数的第一个参数都是self，表示自己这个类

3、





## 其他

1、单引号双引号是一样的

2、-1是最后一个索引，-2是倒数第二这样

3、用index那些都是前闭后开的，所以range(start,end,step)的话，若step为1，那就是从 start到end-1

4、列表中括号，元组是小括号，元组里的值是不能修改的，但是可以对整个元组重新赋值。 字典是大括号

5、def func(\*a) a为列表，可以多个参数传递进去

6、except语句需要一个正确的错误名，或者直接except:

7、yield 类似return 但是返回的是一个迭代器，比如return需要返回一个数组的时候占用空间比较大，用yield就每次只返回一个数

Ex:def xxx(a,b,c):

Yield(b.size,c.size)

调用：for x,y in xxx(D,E,F):

op(x,y)

8、with语句 处理完语句内模块自动调用\_\_exit\_\_()方法

file = open("/tmp/foo.txt")

data = file.read()

file.close()

等于

with open("/tmp/foo.txt") as file:

data = file.read()

1. Print(‘%d....%.3f....%.4f’ %(x1(double),x2(3位小数浮点),x3(4位小数浮点)))
2. b = [x for x in A if x % 2 == 1] 取A中为奇的数

11、set()集合是不允许重复的，可以利用这一特点做临时存储减少时间复杂度

# 基础知识

## 经典问题

1. 进制转换问题

十进制数N和d进制数转换

N=(N div d)\*d+N mod d

即辗转相除，除d得数作为第二次的操作数，除d余数作为最前一位的数。

1. DP相关（见lc5）

DP的递归模式其实是遍历了解的拓扑序，先求解问题再求解子问题（虽然递归调用还是子问题先出来）

DP的自底向上模式其实是遍历了解的拓扑逆序，对任何问题都先求解了其子问题再求解它

1. 回溯法（见lc36）
2. DP不一定是最优的

## 树

1. 树的遍历顺序，先序DLR，中序LDR（正则），后续LRD（逆波兰）
2. 线索二叉树及树的线索化
3. 二叉树相似和等价的定义：相似：二者都为空树或者二者都不为空树，且他们的左右子树分别相似（递归定义），等价是在相似的基础上加一条对应元素相等

1、Merge Two Binary Trees 合并两个二叉树

if(t2==NULL){

return t1;

}

if(t1==NULL){

return t2;

}

t1->val=t1->val+t2->val;

t1->left = mergeTrees(t1->left, t2->left);

t1->right = mergeTrees(t1->right, t2->right);

return t1;

2、Range Sum of BST Problem938 求某个二叉树在L和R范围内所有元素之和

sum=0

if root==None:

return 0

if root.val>=L:

sum += self.rangeSumBST(root.left,L,R)

if root.val<=R:

sum += self.rangeSumBST(root.right, L, R)

if L<=root.val and root.val<=R:

sum+=root.val

return sum

## 列表

1. hashmap可以用set()来实现，用于各种要求重复不重复元素的场景下

## 图

1. 连通图的概念是无向图，**强**连通图的概念是有向图
2. BFS与DFS

DFS：从图中某个定点v出发，访问此顶点，然后依次从v的未被访问的邻接点出发做DFS

BFS：从图中某顶点v出发，访问了v之后依次访问v的各个未曾访问过的邻接点，然后分别**从这些邻接点出发**依次访问他们的邻接点

看成一个队列，DFS是ABCDEF，访问完AB以后，B的子树插入B的位置，BFS是B的子树放到队列末尾

O（n^2）

1. 最小生成树 n点e边

普里姆算法：按点选择，分割点集，寻找连接两点集的最小代价边产生新的点集，O(n^2)

克鲁斯卡尔算法：按边选择，选择代价最小又连接两个连通分量的边产生新的点集，O(eloge)

1. 拓扑排序：偏序（仅部分成员可比较）->全序（全部成员可比较）
2. 选一个没有前驱的节点输出（有多个就是不能比较的那些了）
3. 从图中删除该点及该点发出的边
4. 关键路径：从开始点到完成点最长路径的长度，前后遍历，e(i)==l(i)
5. 最短路径：迪杰斯特拉算法O(n^2)

每次循环选取一个当前最短路径上的所有点都在最短路径点集内的点加入最短路径点集

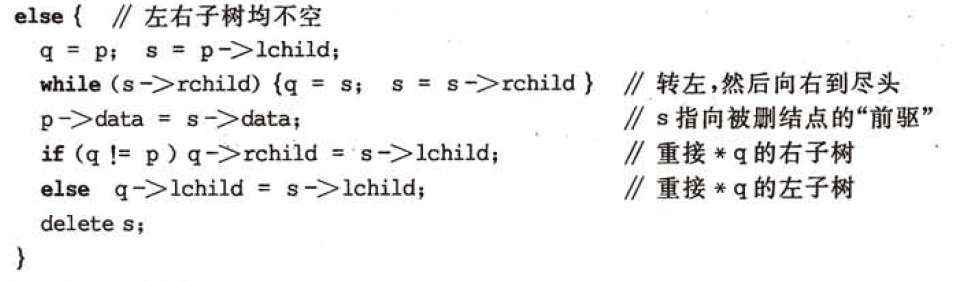
## 查找

1. 二分查找O(logn)
2. 二叉排序树

左小于根小于右，递归搜索：大于根在右找小于在左找

性质：中序遍历可以得到有序序列

重点：二叉排序树的删除

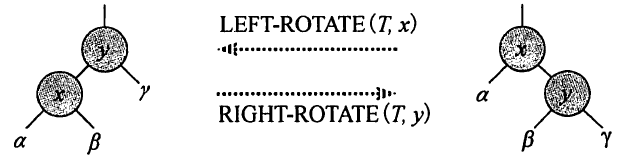


向左再向右到尽头是找到p的直接前驱，用这个点代替p就完成了删除操作。

或者向右再向最左也可以，就是直接前驱或者直接后继来代替p，如果代替点有子节点，那么，删除代替点，并用其子节点接上这个缺口；如果无子节点，则直接删除

1. 平衡二叉树（BST）的旋转

左旋右旋



(设x为父的左孩子，不失一般性）

ex:左旋：x->parent->left=y

y->parent=x->parent（解决父节点问题）

y->left->parent=x

x->right=y->left

y->left=x

x->parent=y（解决内部问题）

总结，每条边需要操作两次，用来旋转的轴点下降

4、b+树b-树 红黑树

红黑树详细：

一、规则（5条性质，based BST）

1、节点颜色为红/黑

2、根节点为黑

3、叶节点为黑

4、红节点的孩子必为黑

5、每个节点到各个叶节点的路径上都包含相同的黑节点

二、性质

1、高度至多为2log2(n+1)

三、插入

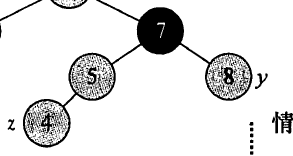
插入只有可能破坏条件2（插入空树）和4。

1、通过二叉搜索树的性质找到插入点，插入为红

2、若插入点的父节点为黑则不破坏条件4，若为红则循环判断

if（父为红）

if（叔为红）

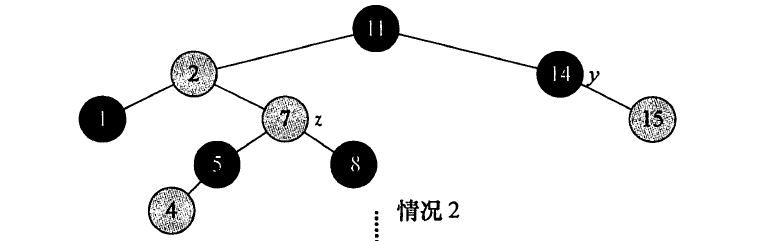


1. 叔、父变黑
2. 爷爷变红
3. 指针指向爷爷，继续外层循环

（这种情况很简单，只要把叔叔、父亲一辈的全部变黑，再爷爷变红，再递推上去就可以了）

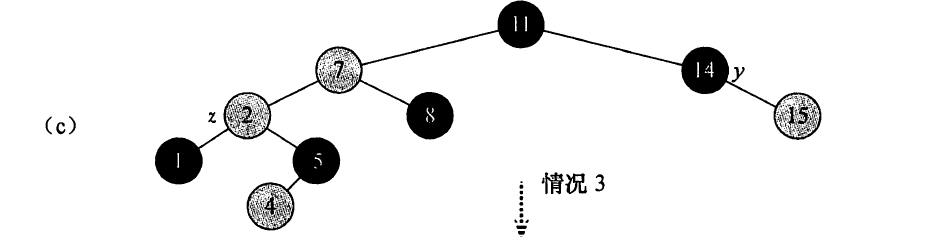
elif （叔为黑）

if（自己是父的右孩子且叔为黑）



在父节点执行左旋

elif（自己是父的左孩子且叔为黑）



父变黑

爷爷变红

右旋

（上面两个操作其实是让右边黑高变短了而左边不变，右旋就可以重新平衡）

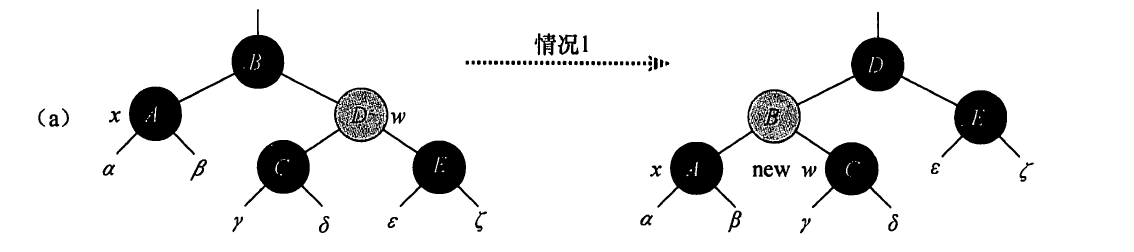
1. 删除（NMSL）
2. 先按BST的删除删掉原节点，然后将替补节点补过来，维持原颜色

根据代替节点的颜色判断，若是黑的，则有可能违背红黑树的性质

若是用节点的后继删除，则从代替节点（被删除节点的后继）的right开始判断，记为x

1. 不失一般性，假设x是其父亲的左孩子

if（x的兄弟为红）



（转换情况，现在x的兄弟为黑了）

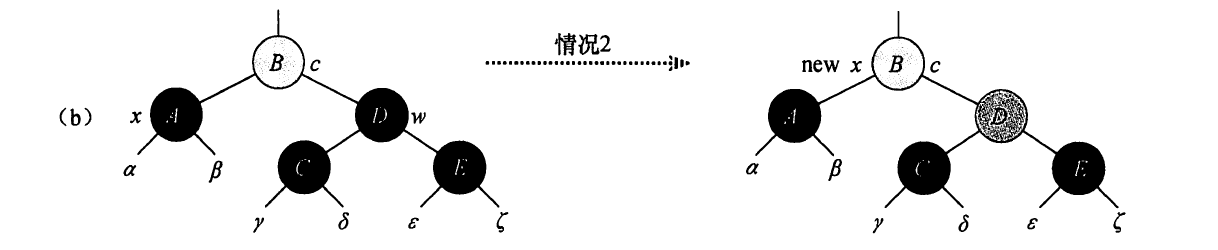
兄弟变黑

父变红

以父为基点左旋

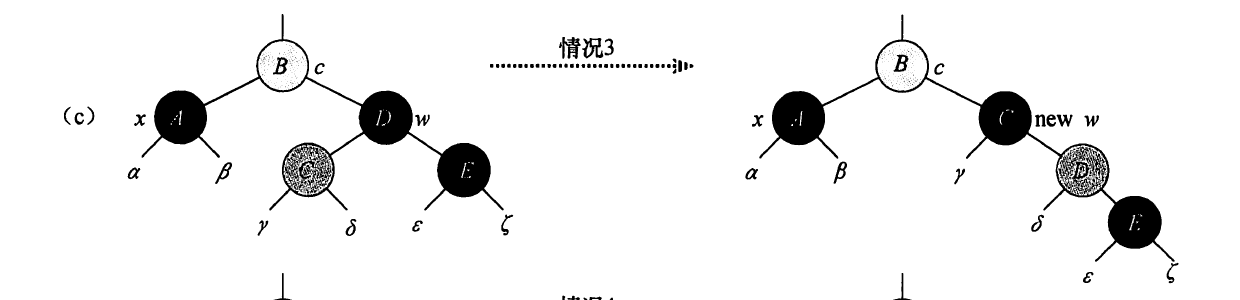
条件1：x兄弟为黑

if（x兄弟为黑，且兄弟的左右孩子都是黑色的）

 兄弟变红

父为新的x

elif（x兄弟为黑，且兄弟的右孩子是黑的（左孩子红））

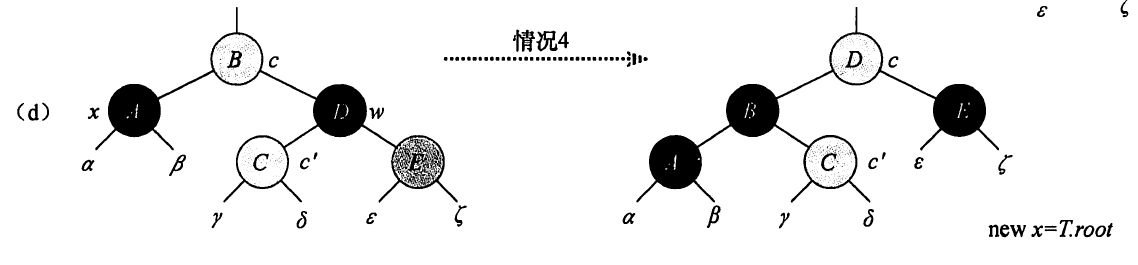


兄弟的左孩子变黑

兄弟变红

以兄弟为轴右旋（现在x兄弟的右孩子红了）

条件2：x的兄弟的右孩子为红



兄弟等于父亲的颜色

父变黑

兄弟的右孩子变黑

父亲为基点左旋

总结：NMSL

## 排序

插排O(n^2)

冒泡O(n^2)

选择排序O(n^2) 不稳定

希尔排序（多份插入排）O(n^2)不稳定

快排O(nlogn)不稳定

归并 nlogn稳定

堆排 nlogn不稳定

堆排序堆的维护：以堆中最后一个元素替代，然后通过元素交换进行调整。堆建立的时候从最后一个非终端节点开始调整即floor(n/2)，其他叶子节点不用管，因为叶子节点满足堆性质

## OS

1. 进程PV操作
2. 生产者消费者问题PAGE60
3. 银行家问题（死锁）
4. 页面置换(LRU)最久未访问

## 二进制类

1. a异或b可以得到a和b所有二进制位的不同，同样的，任何int和0异或都得原来的数

同时，异或满足交换和结合律，所以在一个list里求那个独特的数（其他数都出现偶数次），直接全部异或就vans了

# Leetcodes

1. Q：给定一个list，给定一个数a，找list中的两个数之和等于a

A：一次遍历：for i in list，将a-i保存至set，对后面的数判断是否在set里，O(n)

3、Q：最长不重复子序列

A：用一个字典，记录某个char的最大坐标，同时维护当前最长不重复子序列，当遇到同一个char重复出现时，判断当前的最长子序列是否用到了这个char，如果用到了就更新序列和length。

4、Q:两个有序list，求出中位数，要求log(n)

A：扩展为求第k小的数。求第k小的数，每次可以在两个list中各取第k/2个数，那么对于第k/2个数较小的那个list来说，这之前的数都不可能成为要找的数

注意，只有两种情况，1，k/2小于短的那个list：ex:k=4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |  |  |  |
| 1 | 2 | 4 | 5 | 6 | 7 | 8 | 9 |

2,k/2大于短的那个list，这时候短的list全取 ex:k=12

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |  |  |  |
| 1 | 2 | 3 | 5 | 6 | 7 | 8 | 9 |

红的部分表示剪掉

1. 最长回文子串，动态规划

构造二维数组以在dp(i,j)存储从下标i到j的最长回文子串

由于i<=j所以只用计算矩阵的一半多一根

（1）首先对dp的对角线元素初始化为1，也就是当i==j时，dp[i][j]=1。

这很显然，每个单独的字符其实就是个长度为1的回文串。

（2）当j-i==1：

若s[i]==s[j]，则dp[i][j]=2

当j-i==1时，若s[i]==s[j]，则s[i]和s[j]可以组成一个长度为2的回文串。

（3）当j-i>=2：

若s[i]==s[j]：若dp[i+1][j-1]>0，则dp[i][j]= dp[i + 1][j - 1] + 2;否则dp[i][j]维持;

若s[i]！=s[j]：dp[i][j]维持。

7、9、Q：数字按十进制位reverse。

A：1、可以用int（）和str（）方法互转(py)

2、新数取10的余自乘10，原数整数除10，原=0，新=原

10、44、正则表达式，动态规划

Q:点代表任何char ，星和之前的一个char配对，可以表示0~任意数量的该char

1, If p.charAt(j) == s.charAt(i) : dp[i][j] = dp[i-1][j-1];

2, If p.charAt(j) == '.' : dp[i][j] = dp[i-1][j-1];

3, If p.charAt(j) == '\*':

if p.charAt(j-1) != s.charAt(i) : dp[i][j] = dp[i][j-2] //in this case, a\* only counts as empty

if p.charAt(i-1) == s.charAt(i) or p.charAt(i-1) == '.':

dp[i][j] = dp[i-1][j] //in this case, a\* counts as multiple a

or dp[i][j] = dp[i][j-1] // in this case, a\* counts as single a

or dp[i][j] = dp[i][j-2] // in this case, a\* counts as empty

15、给一个数组，求三个数之和等于0

1、先排序

2、三个指针，最左边一个i for从0循环到末尾，中间一个i+1，最右边一个最右。

根据三个数的和来决定移动哪个指针（最左边的不动）

其实可以得出解决SUM问题的两个思路，1是利用空间保存sum-i 2是先排序利用大小来减少复杂度，但是注意2的方法中的边界条件十分复杂，尤其是4sum问题中。比如4个指针中最小一个指针每次移动的时候判定若该数和前一个数一样直接continue（条件i>=1），次小指针条件j>=i+2

17题输出数字对应的所有字符串

19一次遍历删除链表中倒数第n个节点？

A：用两个指针，第一个先前进n，第二个在头，再共同前进，第一个到末尾第二个就到了要删除的位置

1. 产生符合规则的一对括号，可以用递归做，m和n指示左和右可以产生的数量，一开始左=k右=0，递归产生
2. 按k个一组reverse链表

A：先找到尾部，最后那一组不够长不用处理。

对前面的处理：例如k=5那么12345这一组需要reverse，可以循环4次，第一次是12来reverse，第二次操作可以把已经reverse的部分（这里是12）看成一个整体，只要改这一个整体的头指针和尾指针就可以了，使用三个指针。A指向该组的前一个node（对第一组可以新建一个），B指向当前已经reverse的部分的头，C指向当前已经reverse部分的尾巴，注意操作完以后要把C的next指向下一个node。

1. 除法 重点是不用单纯的一次一次简单减，可以通过移位一次减去2的n次方
2. 找一个vector中所有word无序组成的substring

可以用两个set 一个是存每个word和对应次数，另一个存目前这个已经出现的word和次数

1. 找字典序的下一个序

（1）从右向左找到第一个左临小于右临的位置，左临的位置标记为i

（2）从右向左找到第一个大于a[i] 的数值，位置为j

（3）交换a[i] 与a[j] 的值

（4）将i 右边的部分排成正序

32、求一个字符串中符合条件的最长括号串

这类问题可以DP，重点在于找状态转移方程。DP表保存以i为结尾的最长括号串的长度，则有

1. i左括号，i+1右括号 有DP[i+1]=DP[i-1]
2. i-1右括号 i右括号，若i的右括号可以找到匹配，则有

dp[i]=dp[i - 1] + 2 + dp[i - 1 - dp[i - 1] + 1 - 2] 就是那个可能是多出来的左括号可以和之前的连起来了，三部分，第一部分是i-1的长度，第二部分是2就是两个括号，第三部分是之前的可以和这个连起来

33、循环排序的数组二分查找

1、找到那个最小值

2、根据这个最小值的位置做二分查找，实际的中点位置可以根据目标中点和最小值的位置相加对length取模，看成一个循环数组。

36、37、数独（八皇后），回溯法

1. 函数参数中带有循环终止条件n
2. 验证约束，尝试赋值，调用n+1，回退
3. 满足终止条件，return
4. 求缺少的第一个正数，O（n）的解法

利用下标，将n存储在A[n-1]处，具体实现：if nums[i] 不等于 nums[nums[i]-1]

则swap这两个。这个条件和nums[i]不等于i+1相比，还多了一个检测用于交换的两个数是否相等。

1. 接雨水



分析发现，该点的水位等于该点左右最高水位的较小值。

left[i] = max(left[i - 1], height[i - 1]);

right[i] = max(right[i + 1], height[i + 1]);

level[i] = min(left[i], right[i]);

43、44、全排列：递归回溯

1. 跳跃问题，DP不一定是最优的（n^2），贪心可以n

48、矩阵旋转，求矩阵旋转，如顺时针转90度等于转置再求对称

50、求pow(x,y)

很简单就可以实现，算法的复杂度是O（N），那么有什么办法可以优化呢，现在是O（N），那么就要朝着O（logN）的方向进行优化，以25为例X^25，可以先考虑X^1， X^2 ，X^4， X^8， X^16，然而还差X^9，同理我们将9分解为X^1， X^2 ，X^4， X^8，余下的就是X^1，将每个步骤的最大次方提取出来就是X^16，X^8，X^1，其实就是将25分解成16，8，1也就是25的二进制表示11001。把25分解后如何处理呢，遍历25的每一位i，如果是1，那么result \*= X(2^i)，最后也就求出X^25 = X^1 \* X^8 \*X^16。